Migrating From Cori to Perlmutter: CPU Codes

March 10, 2023

Erik Palmer, Helen He
User Engagement Group
Outline

● Modules
● Programing Environments
● Makefiles and CMake tips
● Example Code Compilation
● Job Scripts & Affinity
Modules: Loading Preinstalled Software
module avail spider
## Modules Loaded by Default:

```
epalmer@perlmutter:login21:~> $ module list

1) craype-x86-milan
2) libfabric/1.15.2.0
3) craype-network-ofi
5) xpmem/2.5.2-2.4_3.30__gd0f7937.shasta
11) PrgEnv-gnu/8.3.3
8) cray-dsmml/0.2.2

7) cray-libsci/23.02.1.1
9) cray-mpich/8.1.24
7) craype/2.7.19
6) gcc/11.2.0
4) perftools-base/23.02.0
12) cpe/23.02

13) xalt/2.10.2
14) Nsight-Compute/2022.1.1
15) Nsight-Systems/2022.2.1
16) cudatoolkit/11.7
17) craype-accel-nvidia80
18) gpu/1.0
```

- CPU Architecture
- Default Programming Environment and Compiler
- GPU Architecture, CUDA-Aware MPI, GPU Profilers

CUDA-Aware MPI by Default!
Default Modules for CPU-only Code

For CPU-only code we recommend:

```sh
module load cpu
```

- CPU Architecture
- Default Programming Environment and Compiler
- Configured for CPU-only MPI

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) craype-x86-milan</td>
</tr>
<tr>
<td>2) libfabric/1.15.2.0</td>
</tr>
<tr>
<td>3) craype-network-ofi</td>
</tr>
<tr>
<td>4) xpmem/2.5.2-2.4_3.30__gd0f7937.shasta</td>
</tr>
<tr>
<td>5) PrgEnv-gnu/8.3.3</td>
</tr>
<tr>
<td>6) cray-dsmml/0.2.2</td>
</tr>
<tr>
<td>7) cray-libsci/23.02.1.1</td>
</tr>
<tr>
<td>8) cray-mpich/8.1.24</td>
</tr>
<tr>
<td>9) craype/2.7.19</td>
</tr>
<tr>
<td>10) gcc/11.2.0</td>
</tr>
<tr>
<td>11) perftools-base/23.02.0</td>
</tr>
<tr>
<td>12) cpe/23.02</td>
</tr>
<tr>
<td>13) xalt/2.10.2</td>
</tr>
<tr>
<td>14) cpu/1.0</td>
</tr>
</tbody>
</table>

CUDA-Aware MPI turned off!
Modules with Lmod

Most Common

- module list
- module load/unload
- module spider
- module swap
- module show

Cool Tricks

- module --redirect spider -r . | grep <string>
- ml -t

More information: man module or https://docs.nersc.gov/environment/lmod/
Hierarchical Structure of Modules on Perlmutter

- Searches all modules without regard for hierarchy
- Displays only modules that can currently be loaded
Module Example: Searching for cray-netcdf

No Longer Recommended

- module avail

*Only shows packages that can be loaded into the current module environment (due to hierarchy) – use module spider instead

More information: https://docs.nersc.gov/environment/lmod/
Module Example: Searching for cray-netcdf

No Longer Recommended

• module avail

*Only shows packages that can be loaded into the current module environment (due to hierarchy) – **use module spider instead**

More information: https://docs.nersc.gov/environment/lmod/
1. module list
2. module load cray-netcdf
3. module show cray-netcdf
4. module avail
5. module spider cray-netcdf
6. module spider cray-netcdf/4.8.1.5
7. module load cray-hdf5
8. module load cray-netcdf
New Hierarchical Structure of Modules on Perlmutter

- Searches for modules without regard for hierarchy
- Displays only modules that can currently be loaded
Loading Modules Modifies Your Environment

module show cray-hdf5

```
family("hdf5")
conflict("PrgEnv-pathscale")
conflict("cray-hdf5")
conflict("cray-hdf5-parallel")
help([[Release info: /opt/cray/pe/hdf5/1.12.2.3/release_info]])
whatis("The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.")
prepend_path("PATH","/opt/cray/pe/hdf5/1.12.2.3/bin")
prepend_path("PKG_CONFIG_PATH","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1/lib/pkgconfig")
setenv("PE_PKGCONFIG_LIBS","hdf5_hl:hfdf5")
setenv("PE_HDF5_PKGCONFIG_LIBS","hdf5_hl:hfdf5")
setenv("PE_FORTRAN_PKGCONFIG_LIBS","hdf5_hl_fortran:hdf5_fortran")
setenv("PE_HDF5_FORTRAN_PKGCONFIG_LIBS","hdf5_hl_fortran:hdf5_fortran")
setenv("PE_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("PE_HDF5_CXX_PKGCONFIG_LIBS","hdf5_hl_cpp:hdf5_cpp")
setenv("CRAY_HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3")
setenv("PE_HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3")
setenv("CRAY_HDF5_VERSION","1.12.2.3")
setenv("CRAY_HDF5_PREFIX","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
setenv("HDF5_DIR","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
setenv("HDF5_ROOT","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1")
prepend_path("CRAY_LD_LIBRARY_PATH","/opt/cray/pe/hdf5/1.12.2.3/gnu/9.1/lib")
prepend_path("MODULEPATH","/opt/cray/pe/lmod/modulefiles/hdf5/gnu/8.0/cray-hdf5/1.12.2")
```
Programming Environments: Configuring Compilers and Linking Libraries
# Programming Environments

## Control Compilers and Libraries on Perlmutter

<table>
<thead>
<tr>
<th>Language</th>
<th>Wrapper</th>
<th><code>PrgEnv-gnu</code> (default)</th>
<th><code>PrgEnv-nvidia</code></th>
<th><code>PrgEnv-cray</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td>CC</td>
<td>g++</td>
<td>nvc++</td>
<td>crayCC (Clang)</td>
</tr>
<tr>
<td>C</td>
<td>cc</td>
<td>gcc</td>
<td>nvc</td>
<td>craycc (Clang)</td>
</tr>
<tr>
<td>Fortran</td>
<td>ftn</td>
<td>gfortran</td>
<td>nvfortran</td>
<td>crayftn</td>
</tr>
<tr>
<td>MPI</td>
<td>-</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
</tr>
</tbody>
</table>

More info: module show `PrgEnv-gnu` module show `nvidia` module show `cce`
## Programming Environments
### Control Compilers and Libraries on Perlmutter

<table>
<thead>
<tr>
<th>Language</th>
<th>Wrapper</th>
<th>PrgEnv-gnu (default)</th>
<th>PrgEnv-nvidia</th>
<th>PrgEnv-cray</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++</td>
<td><strong>CC</strong></td>
<td>g++</td>
<td>nvc++</td>
<td>crayCC (Clang)</td>
</tr>
<tr>
<td>C</td>
<td><strong>cc</strong></td>
<td>gcc</td>
<td>nvc</td>
<td>craycc (Clang)</td>
</tr>
<tr>
<td>Fortran</td>
<td><strong>ftn</strong></td>
<td>gfortran</td>
<td>nvfortran</td>
<td>crayftn</td>
</tr>
<tr>
<td>MPI</td>
<td>-</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
<td>cray-mpich</td>
</tr>
</tbody>
</table>

**More info:**
- module show PrgEnv-gnu
- module show nvidia
- module show cce
Switching Programming Environments

To switch programming environments (PrgEnv) use:

```
module load PrgEnv- *
```

For example, if I am in PrgEnv-gnu and want to switch to PrgEnv-cray, type:

```
module load PrgEnv-cray
```
Suggested Practice: Compiler Wrappers

Cray provides wrappers (CC, cc, and ftn) to the corresponding compilers for each PrgEnv. These wrappers incorporate many flags and features, including cray-mpich, the recommended MPI.

epalmer@nid005015:~/Training> **gcc** helloworld_openmp.c -fopenmp -o hello

epalmer@nid005015:~/Training> **cc** -craype-verbose helloworld_openmp.c -fopenmp -o hello

gcc -march=znver3 -D__CRAY_X86_MILAN -D__CRAY_NVIDIA80 -D__CRAYXT_COMPUTE_LINUX_TARGET -D__TARGET_LINUX__ helloworld_openmp.c -fopenmp -o hello -Wl,-rpath=/opt/cray/pe/gcc-libs -Wl,-Bdynamic -I/opt/cray/pe/mpich/8.1.24/ofi/gnu/9.1/include -I/opt/cray/pe/libsci/23.02.1.1/GNU/9.1/x86_64/include -I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/nvvm/include -I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/extras/CUPTI/include -I/opt/nvidia/hpc_sdk/Linux_x86_64/22.5/cuda/11.7/extras/Debugger/include …(and more)
Automatic Links Provided By The Wrappers

• Depending on modules loaded, compiler wrappers automatically link:

  MPI, LAPACK, BLAS, ScaLAPACK, and more.

• Cray modules, such as cray-hdf5, cray-fftw, etc. are also linked automatically by the compiler wrappers when loaded into the user environment.

Note: Several scientific libraries such as, LAPACK, ScaLAPACK, and BLAS, are included in cray-libsci. For more information use: man libsci.
Modules Link Dynamically by Default

• Many modules prepend the LD_LIBRARY_PATH or CRAY_LD_LIBRARY_PATH, and have their shared libraries dynamically linked.
  
  e.g.  
  ```bash
  module load gsl
  CC gsl_test.cpp -lgsl -lgslcblas -o gsl_test
  ```

• If you are compiling your own shared libraries, consider using the option, `-Wl,-rpath=<library_path>`. Cray wrappers build dynamically linked executables by default.

• On Perlmutter static compilation with `-static` or `CRAYPE_LINK_TYPE=static` can fail and is not supported.

More info: https://docs.nersc.gov/development/compilers/wrappers/
Other Good-to-Know Compiler Settings for CPU

<table>
<thead>
<tr>
<th>GNU</th>
<th>Cray</th>
<th>Nvidia</th>
<th>Description/ Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O0</td>
<td>-O0</td>
<td>-O1</td>
<td>Default Optimization Level</td>
</tr>
<tr>
<td>-Ofast</td>
<td>-Ofast, -flto</td>
<td>-O4, -fast</td>
<td>Aggressive Optimization (some may cause non-bit-identical output)</td>
</tr>
<tr>
<td>-fopenmp</td>
<td>-fopenmp</td>
<td>-mp</td>
<td>Enable OpenMP (not default)</td>
</tr>
<tr>
<td>-g, -O0</td>
<td>-g, -O0</td>
<td>-g (-O0 by default)</td>
<td>Debug</td>
</tr>
<tr>
<td>-v</td>
<td>-v</td>
<td>-v</td>
<td>Verbose</td>
</tr>
</tbody>
</table>

For more information, when in the corresponding PrgEnv type:

```
man gcc/gfortran  man craycc/crayftn  man nvc/nvfortran
```

[https://docs.nersc.gov/development/compilers/](https://docs.nersc.gov/development/compilers/)
Three Quick Tips for Compiling Older Codes

Fortran:

1. For older codes, first try `-fallow-argument-mismatch` then if needed, we recommend the `-std=legacy` flag which includes additional modifications that reduce strictness.

C/C++:

2. Look for flags that reduce strictness, such as `-fpermissive`

3. `-Wpedantic` can warn you about lines that break code standards
CMake and Makefiles: Quick Tips on troubleshooting build systems
Manually Specify Cray Compiler Wrappers

Some build systems such as CMake or Makefiles may be coded to search for CC, CXX and FC environment variables.

In these cases, it is possible to specify the Cray compile wrappers by setting the environment variables in the following way:

\[
\text{CC}=$(\text{which cc}) \quad \text{CXX}=$(\text{which CC}) \quad \text{FC}=$(\text{which ftn})
\]

Or at the configure step,

```
./configure CC=cc CXX=CC FC=ftn F77=ftn
```

More info: https://docs.nersc.gov/development/build-tools/autoconf-make/
https://docs.nersc.gov/development/build-tools/cmake/
Compiling with an Existing Makefile

Suppose you already have a Makefile

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
    $(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
    $(CC) -o hellomake hellomake.o hellofunc.o
```

Makefile from: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
More info: https://docs.nersc.gov/development/build-tools/autoconf-make/

Does not use the Cray Compiler Wrappers!
Compiling with an Existing Makefile

Suppose you have a Makefile

```
CC=gcc
CFLAGS=-I.
DEPS = hellomake.h

%.o: %.c $(DEPS)
  $(CC) -c -o $@ $< $(CFLAGS)

hellomake: hellomake.o hellofunc.o
  $(CC) -o hellomake hellomake.o hellofunc.o
```

Makefile from: https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/
More info: https://docs.nersc.gov/development/build-tools/autoconf-make/
Compiling with an Existing Makefile

Suppose you already have a Makefile

\[
\begin{align*}
\text{CC} &= \text{cc} \\
\text{CFLAGS} &= -I. \\
\text{DEPS} &= \text{hellomake.h} \\
%.o : %.c $(DEPS) \\
& \quad $(\text{CC}) -c -o @$< $(\text{CFLAGS}) \\
\text{hellomake} : \text{hellomake.o hellofunc.o} \\
& \quad $(\text{CC}) -o \text{hellomake} \text{hellomake.o hellofunc.o}
\end{align*}
\]

Makefile from: [https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/](https://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/)
More info: [https://docs.nersc.gov/development/build-tools/autoconf-make/](https://docs.nersc.gov/development/build-tools/autoconf-make/)
CMake: Troubleshooting with ccmake

You can use the in-terminal gui ccmake to troubleshoot CMake builds.

Starting from the directory, I follow a typical CMake build pattern:

```bash
epalmer>$ ls
CMakeLists.txt openmp_helloworld.cpp README.md
epalmer>$ mk build; cd build
epalmer>$ cmake ..
epalmer>$ ls
CMakeCache.txt CMakeFiles cmake_install.cmake Makefile
epalmer>$ ccmake ..
```

This will bring up the ccmake interface.
CMake: Troubleshooting with ccmake

ccmake gui interface (advanced mode):

CMAKE_ADDR2LINE
CMAKE_AR
CMAKE_BUILD_TYPE
CMAKE_COLOR_MAKEFILE
CMAKE_CRAYPE_LINKTYPE
CMAKE_CRAYPE_LOADEDMODULES
CMAKE_CXX_COMPILER

usr/bin/addr2line
/usr/bin/ar
ON
dynamic
craype-x86-milan:libfabric/1.15.2.0:cray
/opt/cray/pe/gcc/11.2.0/bin/gcc

CMAKE_ADDR2LINE: Path to a program.

Keys: [enter] Edit an entry [d] Delete an entry     CMake Version 3.20.4
[l] Show log output [c] Configure
[h] Help         [q] Quit without generating
[t] Toggle advanced mode (currently on)

We can now inspect the options chosen by CMake.

More info: https://docs.nersc.gov/development/build-tools/cmake/
CMake: Troubleshooting with ccmake

ccmake gui interface (advanced mode):

Does not use the Cray Compiler Wrappers!

More info: https://docs.nersc.gov/development/build-tools/cmake/
CMake: Troubleshooting with ccmake

Inspecting another build or possibility:

More info: https://docs.nersc.gov/development/build-tools/cmake/
CMake: Troubleshooting with ccmake

Inspecting another build or possibility:

```
CMAKE_ADDR2LINE
CMAKE_AR
CMAKE_BUILD_TYPE
CMAKE_COLOR_MAKEFILE
CMAKE_CRAYPE_LINKTYPE
CMAKE_CRAYPE_LOADEMODULES
CMAKE_CXX_COMPILER

CMAKE_ADDR2LINE: Path to a program.
Keys: [enter] Edit an entry  [d] Delete an entry  CMake Version 3.20.4
[l] Show log output  [c] Configure
[h] Help  [q] Quit without generating
[t] Toggle advanced mode (currently on)
```

More info:  https://docs.nersc.gov/development/build-tools/cmake/

Uses Cray Compiler Wrappers!
Manually Specify Cray Compiler Wrappers

Some build systems such as CMake or Makefiles may be coded to search for CC, CXX and FC environment variables.

In these cases, it is possible to specify the Cray compile wrappers by setting the environment variables in the following way:

\[
\text{CC} = \$(\text{which cc}) \quad \text{CXX} = \$(\text{which CC}) \quad \text{FC} = \$(\text{which ftn})
\]

Or at the configure step,

\[
./\text{configure} \quad \text{CC} = \text{cc} \quad \text{CXX} = \text{CC} \quad \text{FC} = \text{ftn} \quad \text{F77} = \text{ftn}
\]

More info:  
https://docs.nersc.gov/development/build-tools/autoconf-make/  
https://docs.nersc.gov/development/build-tools/cmake/
Building Applications: Cori codes should be recompiled for Perlmutter
Example CPU Code Compile with MPI and OpenMP

```c
int main(int argc, char *argv[]) {
    int numprocs, rank, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];
    int iam = 0, np = 1;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Get_processor_name(processor_name, &namelen);

    #pragma omp parallel default(shared) private(iam, np)
    {
        np = omp_get_num_threads();
        iam = omp_get_thread_num();
        printf("Hello from thread %d out of %d from process
                %d out of %d on %s\n", iam, np, rank, numprocs, processor_name);
    }
    MPI_Finalize();
}
```

Hellohybrid.c – contains both MPI and OpenMP commands.

Example from: https://rcc.uchicago.edu/docs/running-jobs/hybrid/index.html
Compile with MPI and OpenMP (CPU only)

Compile line: `cc hellohybrid.c -fopenmp -o hellohybrid`
Compile with MPI and OpenMP (CPU only)

Compile line: `cc hellohybrid.c -fopenmp -o hellohybrid`
Compile with MPI and OpenMP (CPU only)

Compile line:

```
cc hellohybrid.c -fopenmp -o hellohybrid
```
List of commands from previous slide

1. `ls`
2. `module list`
3. `cc hellohybrid.c -fopenmp -o hellohybrid`
4. `export OMP_NUM_THREADS=2`
5. `export OMP_PROC_BIND=spread`
6. `export OMP_PLACES=threads`
7. `srun -n 4 -c 2 ./hellohybrid`

*Note: #5 in the gif is incorrect, it should be OMP_PROC_BIND=spread not OMP_PROC_BIND=true.*
Manually Specify Include, Library Location and Links

epalmer> # In this example, we will show how to manually include and link libraries during the compile step. The example code I will use, requires

Manually Specify Include, Library Location and Links
List of commands from previous slide

1. ls
2. module list
3. cc hypre_ex.c -o hypre_ex
4. export HYPRE_DIR=/global/u1/e/elvis/NERSC_User_Training/hypre/src/hypre
5. ls $HYPRE_DIR/lib
6. ls $HYPRE_DIR
7. cc hypre_ex.c -I${HYPRE_DIR}/include -L${HYPRE_DIR}/lib -o hypre_ex
8. srun -n 1 -c 2 ./hypre_ex
Understanding Job Parameters
Guiding Example Batch Job Script

#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Understanding Job Parameters:
Hardware: Node, Processor, Physical Core, Logical CPU
## Perlmutter CPU Node Terms

<table>
<thead>
<tr>
<th>From Perlmutter system architecture</th>
<th>This Talk</th>
</tr>
</thead>
<tbody>
<tr>
<td>2x AMD EPYC 7763 (Milan) CPUs</td>
<td>2x AMD EPYC 7763 (Milan) Processors</td>
</tr>
<tr>
<td>64 Cores per CPU</td>
<td>64 Physical Cores per processor</td>
</tr>
<tr>
<td>2 Hyperthreads per core</td>
<td>2 Logical CPUs per physical core</td>
</tr>
<tr>
<td>4 NUMA domains per socket</td>
<td>4 NUMA domains per processor</td>
</tr>
</tbody>
</table>

### Diagram of Perlmutter CPU Node

![Diagram of Perlmutter CPU Node](image-url)
Perlmutter CPU Compute Node

- Node
- Processor
- Physical Core
- *Logical* CPU
Office Building Analogy for Node Architecture

Compute Node(s)

Processor

Physical Cores

Logical CPUs / Hardware Threads
Sample Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Understanding Job Parameters:
Splitting Up Work: MPI Tasks, (OpenMP) Threads
Cargo Analogy for MPI Tasks & OMP Threads

Simulation Code

MPI Task

(OpenMP) Thread
Guiding Example Batch Job Script

```bash
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code
```

Key Terms:
- Node
- MPI Task
- Logical CPU
- Thread
- Physical Core
- Processor

Advanced Terms:
- NUMA Domain
Settings to Address NUMA Performance

- Use `--cpu_bind=cores` when the # of MPI tasks ≤ the # of physical cores
- Use `--cpu_bind=threads` when the # of MPI tasks > the # of physical cores
- In hybrid MPI/OpenMP code, use at least 8 MPI tasks to avoid NUMA penalties when using OpenMP threads
- The value of `-c` should be ≥ the value of `OMP_NUM_THREADS`
- For thread affinity set:
  
  ```
  OMP_PROC_BIND=spread
  OMP_PLACES=threads
  ```
#!/bin/bash

#SBATCH -N 2
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH -t 0:30:00

export OMP_NUM_THREADS=8
export OMP_PLACES=threads
export OMP_PROC_BIND=spread

srun -n 32 -c 16 --cpu_bind=cores ./demo_code

---

**Key Terms:**
- Node
- MPI Task
- *Logical* CPU
- Thread
- Physical Core
- Processor

**Advanced Terms:**
- NUMA Domain
### Compute Nodes Comparison for CPU Affinity

<table>
<thead>
<tr>
<th></th>
<th>Cori Haswell</th>
<th>Cori KNL</th>
<th>Perlmutter CPU</th>
<th>CPU on Perlmutter GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical cores</td>
<td>32</td>
<td>68</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>Logical CPUs per</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>physical core</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical CPUs per</td>
<td>64</td>
<td>272</td>
<td>256</td>
<td>128</td>
</tr>
<tr>
<td>node</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA domains</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>-c value for srun</td>
<td>$2 \cdot \lfloor 32 / \text{t.p.n.} \rfloor$</td>
<td>$4 \cdot \lfloor 68 / \text{t.p.n.} \rfloor$, but recommended</td>
<td>$2 \cdot \lfloor 128 / \text{t.p.n.} \rfloor$</td>
<td>$2 \cdot \lfloor 64 / \text{t.p.n.} \rfloor$</td>
</tr>
</tbody>
</table>

**t.p.n. = Number of MPI tasks per node  \( \lfloor x \rfloor = \text{floor}(x) \)**
Job Scripts
### Cori Haswell

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ --cpu_bind=cores ./demo_code
```

### Perlmutter CPU

```bash
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \ --cpu_bind=cores ./demo_code
```

**Tip**: As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
Sample Batch Job Script – MPI Only

**Cori Haswell**

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores ./demo_code
```

1280/40 = 32

2 \* ⌊32/32⌋ = 2

**Perlmutter CPU**

```bash
#!/bin/bash

#SBATCH -N 10
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \
--cpu_bind=cores ./demo_code
```

1280/10 = 128

2 \* ⌊128/128⌋ = 2

**Tip:** As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
Sample Batch Job Script – MPI Only

**Cori Haswell**

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \--cpu_bind=cores ./demo_code
```

**Perlmutter CPU**

```bash
#!/bin/bash

#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 8 \--cpu_bind=cores ./demo_code
```

*Tip:* As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
Cori Haswell

```bash
#!/bin/bash
#SBATCH -N 40
#SBATCH -C haswell
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 2 \   
--cpu_bind=cores  ./demo_code
```

Perlmutter CPU

```bash
#!/bin/bash
#SBATCH -N 40
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=1

srun -n 1280 -c 8 \   
--cpu_bind=cores  ./demo_code
```

**Tip:** As a best practice for MPI only runs, include `export OMP_NUM_THREADS=1`
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ? --cpu_bind=cores ./demo_code

Question

What should the value of -c be?
Sample Batch Job Script – MPI & OpenMP

```bash
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ? - --cpu_bind=cores ./demo_code
```

**Question**

512 / 32 = 16

2 \cdot \left\lfloor \frac{128}{16} \right\rfloor = 16

16 \geq 8

**Hint**

Considered because using OpenMP
Sample Batch Job Script – MPI & OpenMP

Question

```
#!/bin/bash

#SBATCH -N 32
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J job_name
#SBATCH -t 1:00:00

export OMP_NUM_THREADS=8
export OMP_PROC_BIND=spread
export OMP_PLACES=threads

srun -n 512 -c ?
   --cpu_bind=cores ./demo_code
```

Process

1. MPI tasks / Nodes
   \[ 512 \div 32 = 16 \]

2. Logical CPUs / MPI tasks
   \[ 2 \cdot \lfloor 128 \div 16 \rfloor = 16 \]

3. Check logical CPUs greater than threads
   \[ 16 \geq 8 \]

Considered because using OpenMP
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>#!/bin/bash</td>
<td>#!/bin/bash</td>
</tr>
<tr>
<td>#SBATCH -N 32</td>
<td>#SBATCH -N 32</td>
</tr>
<tr>
<td>#SBATCH -C cpu</td>
<td>#SBATCH -C cpu</td>
</tr>
<tr>
<td>#SBATCH -q regular</td>
<td>#SBATCH -q regular</td>
</tr>
<tr>
<td>#SBATCH -J job_name</td>
<td>#SBATCH -J job_name</td>
</tr>
<tr>
<td>#SBATCH -t 1:00:00</td>
<td>#SBATCH -t 1:00:00</td>
</tr>
<tr>
<td>export OMP_NUM_THREADS=8</td>
<td>export OMP_NUM_THREADS=8</td>
</tr>
<tr>
<td>export OMP_PROC_BIND=spread</td>
<td>export OMP_PROC_BIND=spread</td>
</tr>
<tr>
<td>export OMP_PLACES=threads</td>
<td>export OMP_PLACES=threads</td>
</tr>
<tr>
<td>srun -n 512 -c ? \</td>
<td>srun -n 512 -c 16 \</td>
</tr>
<tr>
<td>--cpu_bind=cores ./demo_code</td>
<td>--cpu_bind=cores ./demo_code</td>
</tr>
</tbody>
</table>

```bash
srun -n 512 -c 16 \  
--cpu_bind=cores ./demo_code
```
NERSC Job Script Generator

https://iris.nersc.gov/jobscript • https://my.nersc.gov/script_generator.php

This tool generates a batch script template which also realizes specific process and thread binding configurations.

Configuration:

**Machine**
Select the machine on which you want to submit your job.

- Perlmutter - CPU

**Application Name**
Specify your application including the full path.

- my_app

**Job Name**
Specify a name for your job.

- test_job1

**Email Address**
Specify your email address to get notified when the job enters a certain state.

- elvis@lbl.gov

**Quality of Service**
Select the QoS you request for your job.

---

Job script:

```
#!/bin/bash

# Base script generated by NERSC Batch Script Generator on
# https://iris.nersc.gov/jobscript

#SBATCH -N 8
#SBATCH -C cpu
#SBATCH -q regular
#SBATCH -J test_job1
#SBATCH --mail-user=elvis@lbl.gov
#SBATCH --mail-type=ALL
#SBATCH -t 0:30:0

# OpenMP settings:
export OMP_NUM_THREADS=1
export OMP_PLACES=threads
export OMP_PROC_BIND=true

#run the application:
srun -n 128 -c 16 --cpu_bind=cores my_app
```

This should be:

```
export OMP_PROC_BIND=spread
```
Key Suggestions:

- Use module spider for comprehensive module search
- Recompile Cori code with PrgEnv-gnu, PrgEnv-cray, or PrgEnv-nvidia
- Use the compiler wrappers
- Recalculate job script parameters for optimal performance
CPU Hands-on Exercises

● Feel free to use some NERSC prepared CPU examples at
  ○ [https://github.com/NERSC/Migrate-to-Perlmutter/tree/main/CPU](https://github.com/NERSC/Migrate-to-Perlmutter/tree/main/CPU)
  ○ or bring your own applications codes today.

● Follow README.first and README for each example
  ○ hello-example: serial and MPI
  ○ matrix-example (C) or jacobi-example (Fortran): hybrid MPI/OpenMP
  ○ xthi-example: affinity
  ○ gsl_test: using package available from E4S stack

● Perlmutter Compute node reservations, 11:30 - 14:30:
  ○ CPU: `#SBATCH --reservation=pm_cpu_mar10 -A ntrain8 -C cpu`
  ○ Existing NERSC users are added to the ntrain2 project to access node reservations
Thanks for your attention!

More questions? Need help? ...  
http://help.nersc.gov/